Integration der domänenspezifischen Sprache Movisa in den nutzerzentrierten Entwicklungsprozess der Useware

Henning Hager*, Stefan Hennig*, Marc Seißler†, Annerose Braune*

*Institut für Automatisierungstechnik
Technische Universität Dresden
{henning.hager|stefan.hennig|annerose.braune}@tu-dresden.de

†Zentrum für Mensch-Maschine-Interaktion
Technische Universität Kaiserslautern
marc.seissler@mv.uni-kl.de

Abstract: Im Rahmen der modellbasierten Entwicklung von Benutzungsschnittstellen wird die domänenspezifische Sprache Movisa zur Entwicklung von Visualisierungen für die Automatisierungstechnik verwendet. Um mit Movisa gebrauchstaugliche Benutzungsschnittstellen zu entwickeln, erfolgt die Integration in den übergeordneten, nutzerzentrierten Entwicklungsprozess der Useware. Bei der Integration werden allgemeine Herausforderungen der modellbasierten Entwicklung von Benutzungsschnittstellen analysiert und mit Mitteln der modellgetriebenen Software-Entwicklung gelöst. Als zentrale Herausforderung wird in diesem Beitrag die Überwindung der Abstraktionslücke zwischen abstrakten und konkreten Modellen betrachtet und mit einer interaktiven Transformation gelöst. Weiterhin wird die Unterstützung von iterativen Vorgehen in modellbasierten Entwicklungsprozessen anhand der Useware und Movisa untersucht und als Lösung das *Persistant Transformation Mapping* vorgeschlagen.

1 Motivation

Die modellbasierte Entwicklung von Benutzungsschnittstellen (MBUID[1]) in Kombination mit dem Vorgehen der modellgetriebenen Softwareentwicklung (MDSD[2]) verspricht ein nachhaltiges Design von Visualisierungslösungen in der industriellen Automatisierungstechnik. Innerhalb der MBUID-Community werden notwendige Modelle entworfen und durch die etablierte Referenzarchitektur CAMELEON klassifiziert. CAMELEON definiert Modelle auf verschiedenen Abstraktionsebenen: (1) Modelle der *Task & Concepts* (T&C) erfassen Aufgaben, die Anwender mit der Benutzungsschnittstelle verrichten. (2) Modelle der *Abstract UI* (AUI) beschreiben abstrakte Interaktionsobjekte unabhängig von Modalitäten [Van05]. (3) Modelle der *Concrete UI* (CUI) unterliegen einer konkreten Modalität und definieren bspw. für graphische Benutzungsschnittstellen das Layout. CUI-Modelle sind unabhängig von der ausführenden Plattform [Van05]. (4) Modelle der *Final UI* (FUI)

[1] Model-based User Interface Development
[2] Model-driven Software Development

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

beschreiben schließlich die Benutzungsschnittstelle, wie sie auf einer konkreten Plattform ausgeführt wird.

Die MDSD stellt Methoden und Werkzeuge bereit, um Modelle in beliebige Zielartefakte zu überführen. Der Kerngedanke ist dabei die Trennung der fachlichen Inhalte von der technischen Realisierung [HKB10]. *Domänenspezifische Sprachen* (DSL[3]) unterstützen diese Zielstellung maßgeblich, indem sie eine Abbildung des Gegenstandsbereichs zu leisten vermögen. Mit der DSL *Movisa*[4] können speziell auf die Anforderungen der industriellen Automatisierungstechnik zugeschnittene Benutzungsschnittstellen auf CUI-Ebene entwickelt werden. Damit werden die funktionalen Eigenschaften der Visualisierung plattformunabhängig spezifiziert und anschließend mittels *Model-To-Text*-Transformationen in ausführbaren Quellcode für verschiedene Plattformen überführt.

Die Entwicklung gebrauchstauglicher Benutzungsschnittstellen verlangt eine Eingliederung dieses Vorgehens in einen übergeordneten Entwicklungsprozess. Der *Useware*-Entwicklungsprozess bindet den Endnutzer frühzeitig in alle Entwurfsphasen ein und bietet damit ein nutzerzentriertes Vorgehen. Die Realisierung der Nutzeranforderungen wird auf allen Ebenen durch Nutzerevaluationen begleitet; die Ergebnisse werden direkt in die Entwicklungsphasen zurückgeführt. Damit erfolgt die Entwicklung iterativ. Abbildung 1 veranschaulicht diesen Prozess.

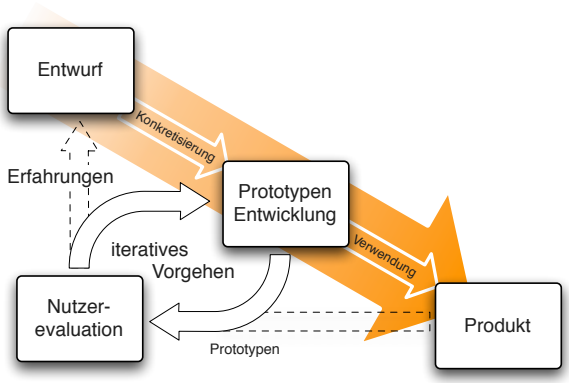


Abbildung 1: Iterative Entwicklung mit paralleler Nutzerevaluation

In diesem Beitrag werden Anforderungen für die Integration der DSL Movisa in den Useware-Entwicklungsprozess erarbeitet sowie allgemeine Herausforderungen der modellbasierten Entwicklung von Benutzungsschnittstellen hervorgehoben. Eine der betrachteten Herausforderungen ist die Überbrückung der Abstraktionslücke bei Transformation abstrakter Modelle (die Modelle der Useware, die Movisa vorgelagert sind) in konkrete Modelle (Movisa-Modelle). Eine weitere Herausforderung stellt die iterative modellbasierte Entwicklung dar, denn im nutzerzentrierten Vorgehen können Änderungen in

[3]Domain Specific Language
[4]Model Driven Development of Visualization Solutions in Industrial Automation

jeder Entwicklungsphase stattfinden; eine mögliche Folge sind Inkonsistenzen zwischen den einzelnen Modellen. Dieser Beitrag schlägt einen allgemeingültigen Ansatz vor, der die Abstraktionslücke mittels *interaktiven Transformationen* schließt und ausgehend davon iterative modellbasierte Entwicklungen durch ein *Persistant Transformation Mapping* (Petmap) erlaubt.

Der Beitrag gliedert sich im Weiteren wie folgt: Abschnitt 2 führt die verwendeten Technologien grundlegend ein. Abschnitt 3 analysiert die Integration von Movisa in den Useware-Entwicklungsprozess und konkretisiert die auftretenden Herausforderungen. Abschnitt 4 stellt den vorgeschlagenen Ansatz vor. Dieser wird anhand einer Fallstudie in Abschnitt 5 demonstriert. Einen Überblick zu verwandten Arbeiten gibt Abschnitt 6. Die Ergebnisse dieses Beitrags werden in Abschnitt 7 diskutiert und zusammengefasst.

2 Grundlagen

Die Integration von Movisa in den Useware-Entwicklungsprozess erfordert einige Grundlagen, die nach Abbildung 2 dem übergeordneten Gebiet der MDSD zugeordnet werden können. Die MDSD liefert die methodischen Grundlagen für Movisa und die modellbasierte Entwicklung von Benutzungsschnittstellen. Die Useware bzw. deren Entwicklungsprozess liegt nicht vollständig in diesen Bereichen, schneidet sie aber, da der Entwicklungsprozess nicht auf die Verwendung von Modellen festgelegt ist.

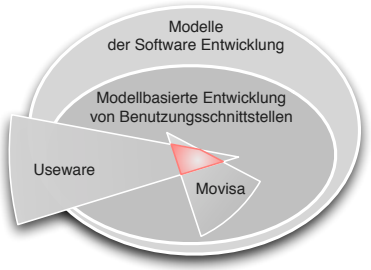


Abbildung 2: Übersicht und Einordnung der verwendeten Technologien

Zunächst werden einige Grundlagen der MDSD bzgl. Transformationen aufgegriffen und im Weiteren wird die DSL Movisa vorgestellt. Die Vorstellung des zu verwendenden nutzerzentrierte Entwicklungsprozess der Useware rundet diesen Abschnitt ab.

2.1 Transformationen in der modellgetriebenen Software-Entwicklung

Für die gegebene Problemstellung der Integration von Movisa in den Useware-Entwicklungsprozess ist eine Transformation von Useware-Modellen (siehe Abschnitt 2.3) zu Movisa (siehe Abschnitt 2.2) notwendig. Im Folgenden werden daher der Aufbau und für den

Beitrag wesentliche Eigenschaften von Transformationen der MDSD vorgestellt.

In der MDSD können *formale* Modelle – Modelle, die einem wiederum formalen Metamodell unterliegen – durch Transformationen ineinander überführt werden. Die Überführung der konkreten Quell- und Zielmodellinstanzen basiert in den Transformationen auf *Mappings*, die allen Elementen des Quell-Metamodells Elemente des Ziel-Metamodells zuordnen (siehe Abbildung 3). Bei der Überführung von Modellen (besonders bei Modellen unterschiedlichen Abstraktionsgrades) können *mehrdeutige* Mappings auftreten, d. h. dass den Elementen des Quellmetamodells verschiedene Elemente des Zielmetamodells zugeordnet werden können. Als Möglichkeiten zur Lösung dieses Zuordnungsproblems stellen Hennig et al. die folgenden Varianten vor [HdBLB11]:

Standardzuordnung: Die Mehrdeutigkeiten werden in der Transformation nicht berücksichtigt, stattdessen werden für mehrdeutige Zuordnungen Standardelemnte verwendet, die u. U. im generierten Modell ausgetauscht werden müssen.

Annotation des Quellmodells: Vor der Transformation wird das Quellmodell mit zusätzlichen Informationen (Annotationen) für das Mapping versehen, sodass die Transformation die annotierten Elemente eindeutig zuordnen kann.

Externe Mappings: Im Sinne der MDSD wird Funktionalität von Technologie getrennt und die Transformation verwendet externe Quellen (z. B. Mapping-Modelle) für die Elementzuordnung.

Interaktive Transformation: Die Transformation ist zur Laufzeit auf den Entwickler angewiesen, der das zu realisierende Mapping spezifiziert. Im Gegensatz zu den anderen Varianten benötigt der Entwickler lediglich Fachwissen der Domäne und nicht spezifisches Wissen über Modellierungstechnologien oder Metamodelle.

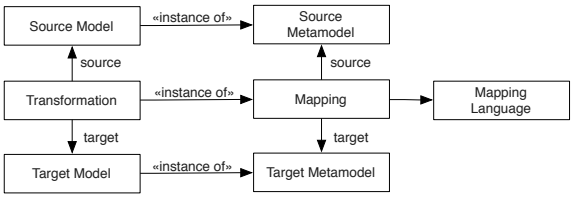


Abbildung 3: Vereinfachte Metamodell-Struktur von Transformationen nach [GPR06]

Weitere wichtige Eigenschaften von Transformationen für die Integration von Movisa in den Useware-Entwicklungsprozess sind:

Grad der Automatisierung: Transformationen sind *automatisch* oder *semi-automatisch* [SS08]. Automatische Transformationen sind selbständig lauffähig; bei semi-automatischen hingegen sind externe Informationen (z. B. Mappings) zur Durchführung der Transformationen notwendig.

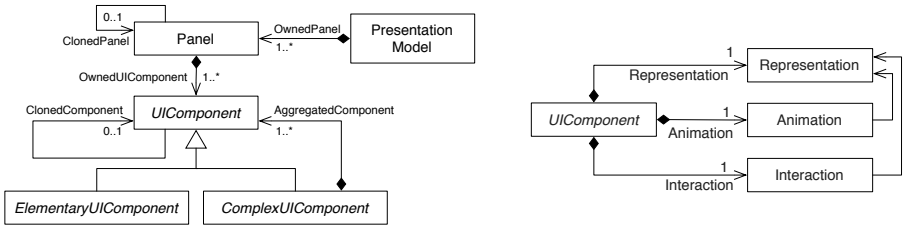
Verfolgbarkeit (Traceability): Die Reproduktion von Transformationen bzw. deren Ergebnisse sind nur möglich, wenn sie *verfolgbar* [CH03] sind, d. h. die Beziehungen zwischen den Quell- und Zielelementen erhalten bleiben.

2.2 Movisa

Wichtige Benutzungsschnittstellen in der Automatisierungstechnik sind u. a. Prozessvisualisierungen zur Überwachung und Steuerung von Anlagen. Für die korrekte Funktionsweise der Visualisierung stellt die Automatisierungstechnik Anforderungen an die Benutzungsschnittstellen, die von üblichen Office-Systemen nicht erfüllt werden. Zum einen werden spezielle Interaktionsobjekte, wie z.B. Trendgraphiken und Analogmeter, für die Interaktion mit dem System benötigt und zum anderen sind weitere funktionale Aspekte der Visualisierung notwendig: die Anbindung von Prozessdaten und die Spezifizierung interner Applikationslogik.

Die Einsatzzeit der Prozessvisualisierungen in automatisierungstechnischen Anlagen ist verglichen mit den Innovationsraten der verwendeten Standard-IT-Komponenten sehr lang. Um die technologischen und vor allem sicherheitsrelevanten Weiterentwicklungen der IT-Komponenten zu berücksichtigen, ist die wiederkehrende Neuentwicklung funktional identischer Benutzungsschnittstellen erforderlich. Zusätzlich wird der Entwicklungsaufwand durch die wachsende Anzahl der zu unterstützenden Plattformen in der Automatisierungstechnik erhöht. Generische Entwicklungsansätze sind wünschenswert und sinnvoll, um die Entwicklung zu vereinfachen und dabei die spezifischen Anforderungen zu erfüllen. Die in der MBUID etablierte Beschreibungssprachen wie UIML[5] [HSL+08] können diese Anforderungen nicht erfüllen [BH07].

Diese Umstände haben die Entwicklung von Movisa [HB11] motiviert. Movisa ist eine speziell für die Domäne der Automatisierungstechnik entworfene DSL, die plattformunabhängige Entwicklung von Benutzungsschnittstellen auf CUI-Ebene ermöglicht. Das Movisa-Metamodell beinhaltet drei Teilmodelle: (a) *Presentation Model* für die Beschreibung der Darstellung, (b) *Client Data Model* zur Anbindung der Visualisierung an Prozessdatenserver und (c) *Algorithm Model*, um beliebige Anwendungslogik (z. B. zur Überprüfung von Nutzereingaben oder Datenaufbereitung der Prozessvariablen) zu realisieren. Für die Integration von Movisa in den Useware-Entwicklungsprozess ist nur das Presentation Model berücksichtigt, da die derzeitigen Werkzeuge des Entwicklungsprozesses keine funktionalen Eigenschaften wie Prozessdatenanbindung oder Anwendungslogik berücksichtigen.



(a) Auszug aus dem Presentation Model (b) Eigenschaften der UIComponent

Abbildung 4: Das Presentation Model zur Darstellungsspezifizierung der Visualisierung.

[5] User Interface Markup Language

Das Presentation Model ist für die graphische Modalität konzipiert und stellt Panels zur Beschreibung von Ansichten und Interaktionsobjekte (*UIComponent*) bereit (siehe Abbildung 4a). Es werden elementare und komplexe UIComponents (*ElementaryUIComponent*, *ComplexUIComponents*) unterschieden. Die elementaren stellen neben Standardelementen wie Buttons, Bilder etc. zusätzlich domänenspezifische Elemente der Automatisierungstechnik wie Trendgraphiken (*Trend*) und Analogmeter (*Gauge*) bereit (siehe Tabelle 1). Mit ComplexUIComponents können UIComponents zu Hierarchien aggregiert und wiederverwendet werden.

Tabelle 1: Übersicht und Funktion der ElementaryUIComponents von Movisa

UIComponent	Beschreibung
AlarmControl	Informiert über Alarme des Prozesses
Button	Ausführung von Funktionen des Visualisierungssystems
CheckBox	Auswahl von beliebigen Werten bzw. Elementen
Gauge	Darstellung von Werten auf einer Skala mit Zeiger
Image	Anzeige von Bildern
Input	Eingabefeld von beliebigen Werten
DropDown	Auswahl eines einzelnen Wertes
RadioButton	Auswahl eines einzelnen Wertes
Slider	Stufenlose Auswahl eines Wertes
TextLabel	Anzeige von Text
Trend	Darstellung eines Wertes in Abhängigkeit der Zeit

Die charakteristischen Eigenschaften aller UIComponents sind in Abbildung 4b dargestellt: (a) *Representation* zur konkreten Darstellung auf dem Bildschirm mit Größe, Breite etc., (b) *Animation* für die Spezifikation von veränderlichen Eigenschaften zur Visualisierung von Prozessdaten (z. B. Höhen- oder Positionsanimation) und (c) *Interaction*, um Benutzerinteraktionen wie Klicken zu modellieren.

2.3 Useware

Die Useware [Zü04] eines Systems umfasst alle Hard- und Software-Komponenten, die dem Nutzer für die Bedienung zur Verfügung stehen. Zur Entwicklung dieser Komponenten wird der nutzerzentrierte Useware-Entwicklungsprozess (siehe Abbildung 5) verwendet, der sich in vier Entwicklungsphasen mit paralleler Evaluation gliedert. In den Entwicklungsphasen wird die Benutzungsschnittstelle auf unterschiedlichen Abstraktionsniveaus beschrieben, deren Beschreibungen im Verlauf des Prozesses weiter konkretisiert werden. Innerhalb der parallel verlaufenden Nutzerevaluation werden die Zwischenergebnisse der Entwicklungsphasen anhand prototypischer Realisierungen auf Erfüllung der Nutzeranforderungen überprüft. Durch unmittelbare Rückführung der Evaluationsergebnisse in die Entwicklungsphasen entstehen Benutzungsschnittstellen mit hoher Gebrauchstauglichkeit.

Im Folgenden werden die Phasen und einige der verwendeten Modelle charakterisiert, damit in Abschnitt 3 eine geeignete Integration von Movisa vorgenommen werden kann. In der *Analyse* wird die Datenerhebung bzgl. der Systemnutzer, deren Aufgaben und Ar-

ISBN 978-3-88579-286-4

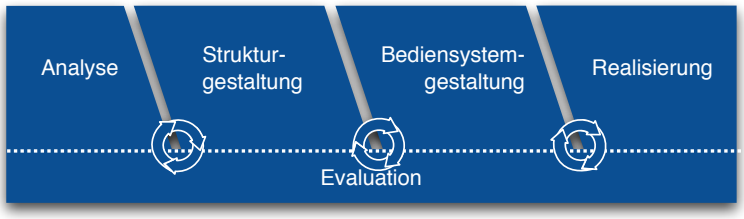


Abbildung 5: Die Phasen des Useware-Entwicklungsprozesses

beitsweisen vorgenommen. Die verwendeten Technologien und Werkzeuge (UseDDL, TAMaRA [MGS08]) der Analyse dienen der Erfassung und Strukturierung der Daten. Alle folgenden Phasen bauen auf der Datenbasis der Analyse auf und leiten eine an die Anforderungen abgestimmte Benutzungsschnittstelle her. Die sich anschließende *Strukturgestaltungsphase* verwendet die Daten zunächst zur Bildung des Benutzungsmodells (UseML[6] [Mei10]), das die erste Struktur der Benutzungsschnittstelle mit Nutzeraufgaben und Temporaloperatoren[7] darstellt. Aus diesem Benutzungsmodell wird des Weiteren in der Strukturgestaltungsphase ein Dialogmodell (DISL[8] [Sch07]) abgeleitet [Sei09]. Der nächste Schritt der Konkretisierung findet in der *Bediensystemgestaltungsphase* statt, in der die Modalität (graphisch, auditiv) und das damit verbundene Layout festgelegt wird. Als Technologie für die graphische Modalität unterstützt die Useware bisher UIML, deren Interaktionselemente jedoch nicht den Anforderungen der Automatisierungstechnik genügen (siehe Abschnitt 2.2). In der letzten Phase des Entwicklungssprozesses, der *Realisierung*, findet die Umsetzung in Quellcode statt. Die für die nutzerzentrierte Entwicklung wesentliche *Evaluation*, die begleitend zu allen Entwicklungsphasen stattfindet, verwendet die jeweiligen Prototypen der verschiedenen Abstraktionsstufen. Alle gesammelten praktischen Erfahrungen und Bewertungen der Benutzer werden in die Entwicklung zurückgeführt. Auf diese Weise erhalten die Entwickler qualitative Auswertungen der entworfenen Benutzungsschnittstelle, mit denen Fehler und mögliche Verbesserungen frühzeitig erkannt werden können.

Der Aufbau und die Elemente des Dialogmodell DISL sind für die Integration von Movisa von zentraler Bedeutung. DISL verwendet zur Beschreibung eine Zustandsmaschine mit Dialogzustände, denen abstrakte, modalitätenunabhängige Interaktionsobjekte (*Widget*; siehe Tabelle 3) zugeordnet werden.

[6] Useware Markup Language

[7] Temporaloperatoren geben die zeitliche Beziehung zur Ausführung der Nutzeraufgaben an; z. B. sequentiell oder parallel.

[8] Dialog and Interface Specification Language

Tabelle 2: Übersicht und Verwendung der DISL-Widgets nach [Mei10]

Widget	Beschreibung
variablefield	Ausgabe einzelner Werte
textfield	Textausgabe
variablebox	einfache Werteeingabe
textbox	Texteingabe
command	Ausführung einer Aktion/Funktion
confirmation	Ausführung einer Aktion/Funktion nach Nutzerbestätigung
choicegroup	Gruppierung und Auswahl von Widgets
widgetlist	Reine Gruppierung von Widgets

3 Analyse der Integration und ihrer Herausforderungen

Nach Meixr N
Referenzarc n-
same Basis uf
CUI-Ebene e-
diensystemg

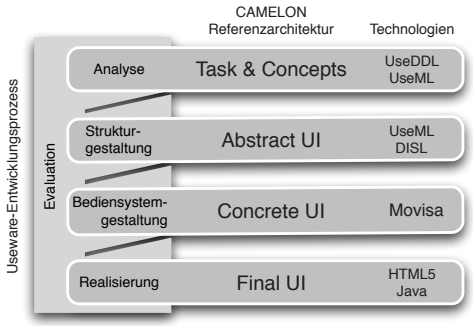


Abbildung 6: Einordnung der Useware und der DSL Movisa in die CAMELEON Referenzarchitektur

Aus diesem Grund basiert die Transformation zur Erzeugung des Movisa-Modells auf dem DISL-Dialog-Modell der Strukturgestaltungsphase. Innerhalb der DISL-Movisa-Transformation müssen die DISL-Widgets auf die UIComponents von Movisa abgebildet werden. Tabelle 3 stellt die Mappings der Elemente dar: Nur das Gruppierungs-Widget widgetlist lässt sich eindeutig einer UIComponent (ComplexUIComponent) zuordnen; bei den übrigen Elementen liegt ein Mapping-Problem vor: Die Zuordnungen zwischen den DISL- und Movisa-Elementen sind nicht eindeutig.

Die Transformation muss das iterative Vorgehen der Useware-Entwicklung unterstützen, sodass nachträgliche Veränderungen im DISL-Modell (z. B. aufgrund von Ergebnissen aus der Evaluation) im Movisa-Modell berücksichtigt werden. Bei wiederholter Transformation müssen die spezifischen Elementzuordnungen reproduziert werden, um eine konsistente und durchgängige Entwicklung zu sichern.

Tabelle 3: Zuordnung der DISL- und Movisa-Elemente.

Movisa UICompsonents	DISL Widgets: variablefield	textfield	variablebox	textbox	command	confirmation	choicegroup	widgetlist
AlarmControl	✔	✔			✔	✔		
Button	✔	✔			✔	✔		
CheckBox	✔	✔			✔	✔	✔	
Gauge	✔	✔			✔	✔		
Image	✔	✔			✔	✔		
Input	✔	✔	✔	✔	✔	✔		
DropDown	✔	✔			✔	✔	✔	
RadioButton	✔	✔			✔	✔	✔	
Slider	✔	✔			✔	✔		
TextLabel	✔	✔			✔	✔		
Trend	✔	✔			✔	✔		
ComplexUIComponent	✔	✔	✔	✔	✔	✔	✔	✔

4 Realisierung der Integration

Zur Lösung des Mapping-Problems wurden in Abschnitt 2.1 vier Methoden vorgestellt. Die Verwendung von Standardmappings erweist sich für den Entwicklungsprozess als hinderlich, da nach jeder Transformation manuelle Nacharbeiten am erzeugten Modell notwendig sind. Auch die Annotierung der Modellelemente zur Spezifizierung der Mappings ist nicht sinnvoll, da Eingriffe auf (Meta-) Modellebene notwendig sind, die spezifisches Fachwissen der Metamodelle erfordern. Die übrigen Methoden – interaktive Transformation und externe Mappings – werden bei der Integration von Movisa in den Useware-Entwicklungsprozess genutzt. Die interaktive Transformation fordert den Entwickler auf für jedes Widget, das nach Tabelle 3 über kein eindeutiges Mapping verfügt, die zu erzeugenden Movisa-UIComponent zu wählen. Dabei werden die Optionen nach Tabelle 3 eingeschränkt, sodass nur gültige Mappings ausgewählt werden können.

Zur Unterstützung der iterativen Entwicklung ist die Transformation verfolgbar (siehe Abschnitt 2.1). Auf diese Weise können die bereits in den vorangegangenen Iterationen bzw. Transformationen spezifizierten Mappings wiederverwendet werden und der Entwickler muss nur Mappings für neu hinzugekommene Elemente festlegen. Die Verfolgbarkeit wird durch das *Persistant Transformation Mapping* realisiert, das gewählte Mappings in einem Modell – der Petmap – persistiert. Nach Interaktion mit dem Entwickler zur Elementauswahl, wird in der Petmap ein neues Mapping (siehe Abbildung 7) der konkreten Elemente angelegt. Bei erneuten Transformationen dient die Petmap als externes Mapping-Modell und die Transformation erfolgt für bekannte Elemente ohne Entwicklerinteraktion.

Die Petmap ist nicht auf die Modelle der Useware oder Movisa festgelegt, es lassen sich Mappings zwischen Elementen aus beliebigen Metamodellen beschreiben. Der Aufbau gliedert sich dazu in drei Teile: (1) Im *MetaModel Repository* werden alle Metamodelle und ihre Elementtypen referenziert, die zur Typisierung der Modelle bzw. Modellelemente verwendet werden. (2) Das *Model Repository* beinhaltet alle Quell- und Zielmodelle und

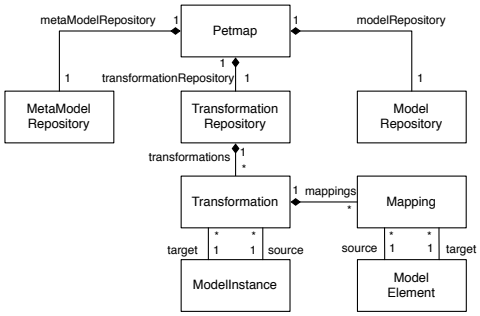


Abbildung 7: Auszug aus dem Petmap-Metamodell. Transformationen zeigen auf Quell- und Zielmodellinstanzen und beinhalten Mappings.

deren Elementen, die in den Mappings referenziert werden werden. (3) Das *Transformation Repository* fasst alle durchgeführten Transformationen mit ihren Mappings zusammen, sodass alle Ergebnisse reproduziert werden können.

Die Realisierung der Integration erfolgt durch die interaktive Transformation von DISL nach Movisa unter Verwendung der Petmap. Abbildung 8 fasst das Vorgehen zusammen. Bei erstmaliger Transformation (①) sind für alle mehrdeutigen Elemente des DISL-Modells Entwicklerinteraktionen notwendig, die in der Petmap gespeichert werden. Innerhalb einer weiteren Entwicklungsiteration werden Veränderungen im DISL-Modell (②) vorgenommen und durch eine erneute Transformation (③) in das Movisa Modell weitergereicht. Für die erneute Transformation werden die gespeicherten Mappings aus der Petmap wiederverwendet und die Entwicklerinteraktionen reduzieren sich auf neue DISL-Elemente. Als Ergebnis entsteht ein Replikat des Modells mit Berücksichtigung der Veränderungen aus dem DISL-Modell.

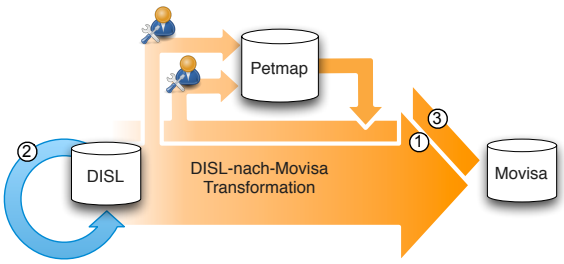


Abbildung 8: Verwendung der interaktiven Transformation und der Petmap

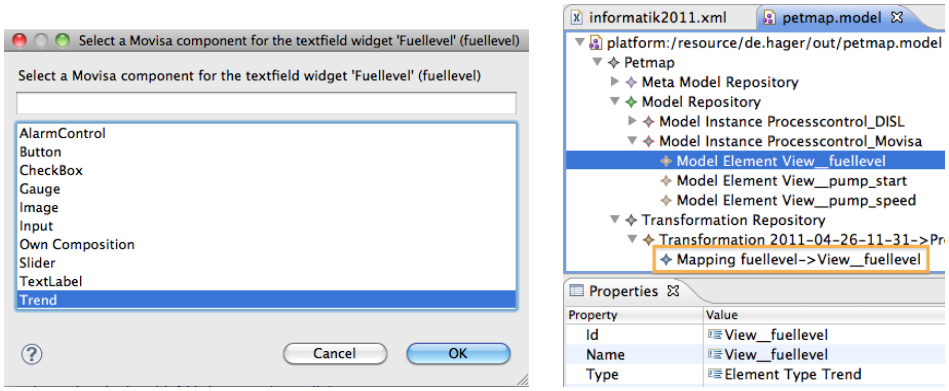
5 Fallstudie

Die vorgenommene Integration von Movisa in den Useware-Entwicklungsprozess wird anhand einer Fallstudie demonstriert. Grundlage ist die Überwachung und Steuerung des Füllstands eines Behälters. Der Auszug des DISL-Modells in Abbildung 1 zeigt die drei Widgets (siehe Tabelle 2) des Zustandes `View`: (1) `fuellevel`: Darstellung der aktuellen Füllstandshöhe des Behälters, (2) `pump_start`: Startet das Abpumpen des Behälters, (3) `pump_speed`: Darstellung der aktuellen Pumpendrehzahl.

Listing 1: Auszug des DISL-Modells

```
<interface id="View" state="start">
        <structure>
          <widget generic-widget="textfield" id="fuellevel"/>
          <widget generic-widget="command" id="pump_start"/>
          <widget generic-widget="textfield" id="pump_speed"/>
        </structure>
  <style>    <!-- ... --> </style>
  <behavior> <!-- ... --> </behavior>
</interface>
```

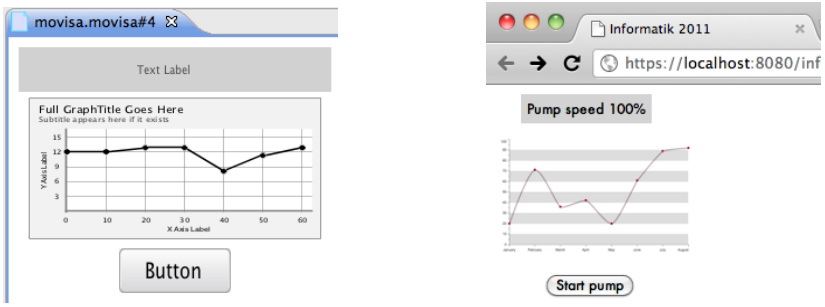
Während der DISL-nach-Movisa-Transformation wird der Entwickler zur Auswahl der Zielelemente für alle Widgets aufgefordert (siehe Abbildung 9a), da nach Tabelle 3 für keines der Elemente eindeutige Mappings existieren. Die Mappings werden in der Petmap abgespeichert, sodass sie bei erneuter Transformation zur Verfügung stehen. Aus der `style`-Beschreibung (siehe Listing 1) werden Darstellungeigenschaften (z. B. Text des Buttons) und aus `behavior` Navigationseigenschaften für das Movisa-Modell abgeleitet. Abbildung 9b zeigt das Mapping in der Petmap für das DISL-Widget `fuellevel` zu der Movisa-UIComponent `View_fuellevel` vom Typ Trend. Das erzeugte Movisa-Modell ist in Abbildung 10a dargestellt und die daraus erzeugte HTML/JavaScript-Lösung der Visualisierung in 10b.



(a) Dialog zur Auswahl des Mappings

(b) Erzeugtes Mapping in der Petmap

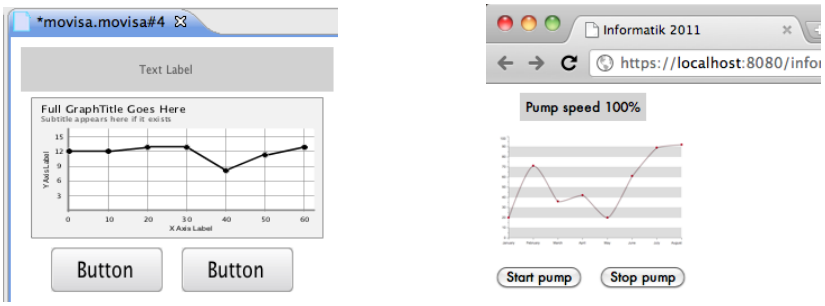
Abbildung 9: Auswahl (a) und Speicherung (b) des Mappings für `fuellevel`.



(a) Erzeugtes Movisa-Modell (b) Resultierende Visualisierung

Abbildung 10: Die Visualisierung als Movisa-Modell (a) und Weblösung (b)

Im Verlauf der weiteren Entwicklung und Evaluation des DISL-Modells stellt sich heraus, dass die Funktion zum Stoppen der Pumpe fehlt. Das zusätzliche command-Widget wird im DISL-Modell ergänzt und durch eine erneute Transformation im Movisa-Modell berücksichtigt. Für diese Transformation wird die Petmap als Mapping-Modell verwendet und somit muss der Entwickler nur für das neu hinzugekommene command-Widget eine UIComponent spezifizieren. Abbildung 11 zeigt die endgültige Version der erzeugten Visualisierung zur Steuerung und Beobachtung des Behälters. Diese Lösung ist das Resultat des nutzerzentrierten Entwicklungsprozesses. Damit verspricht diese Lösung einen effizienten Einsatz der Benutzungsschnittstelle, die Fehlbedienungen auf ein Mindestmaß reduziert.



(a) Movisa-Modell nach erneuter Transformation (b) Resultierende Visualisierung

Abbildung 11: Finales Movisa-Modell (a) und Weblösung (b)

6 Verwandte Arbeiten

Das Problem der Überbrückung der Abstraktionslücke von abstrakten zu konkreten Modellen ist eine Problemstellung der MBUID, das erstmals durch Puerta und Eisenstein [PE99] untersucht wurde. Als Entwicklungsprozess stellt Puerta MOBI-D vor, das den Entwickler bei der Konkretisierung der Benutzungsschnittstelle beliebige Mappings spezifizieren lässt. Dadurch erhält der Entwickler große Freiheiten zur Individualisierung, andererseits wird er unzureichend bei der Entwicklung unterstützt, die außerdem wenig automatisiert ist. Bei der Reproduzierbarkeit der durchgeführten Transformation wird der Entwickler nicht unterstützt, da erneute Transformationen wieder frei wählbare Mappings benötigen. Eine weitere Entwicklungsumgebung ist MARIA [PSS09], die über ein graphisches Tool verfügt, mit dem Entwickler die Mappings der Elementen spezifizieren kann. Bei diesen Mappings können keine mehrdeutigen Zuordnungen verwendet werden, sodass Anpassungen im erzeugten Modell vorgenommen werden müssen. Das iterative Entwickeln ist in MARIA nicht berücksichtigt. DynaMo-AID [CLC04] stellt Standardmappings für die automatische Konkretisierung bereit und erfordert somit ebenfalls die Nachbearbeitung der Modelle. Auch in DynaMo-AID ist die iterative Entwicklung nicht vorgesehen. Aquino verwendet Transformationsprofile und lagert die Mappings in ein eigenes Modell aus [Aqu09]. Die im Vorfeld für alle auftretenden Elemente spezifizierten Mappings werden von Semi-automatische Transformationen zur Erzeugung der Zielmodelle verwendet. Die Mappings sind regelbasiert beschrieben und erlauben keine individuelle Anpassung während der Transformation. Durch die externen Mappings lassen sich die Transformationsprofile für die iterative Entwicklung nutzen. Ein weiterer Ansatz [Ran10] ist die interaktive Entwicklung von Benutzungsschnittstellen mit Verwendung regelbasierter Transformationen. Die Regeln und Mappings werden von einem Editor angepasst, wenn der Entwickler das erzeugte Modell nachbearbeitet. Bei iterativer Entwicklung ist auch bei diesem Ansatz die manuelle Nachbearbeitung des erzeugten Modells notwendig, um die gewünschten Ergebnisse zu erhalten. In UsiXML [LV04] lassen sich innerhalb des Metamodells Transformationen und Mappings festlegen, die durch bereitgestellte Werkzeuge bearbeitet werden können. Dieses Vorgehen erwartet detaillierte Kenntnisse des UsiXML-Metamodells und manuelle Spezifikation der Mappings vor der Transformation.

Der von uns vorgeschlagene Ansatz erfordert, im Gegensatz zu den genannten Arbeiten, kein detailliertes Fachwissen des Entwicklers über (Meta-) Modelle und keine Korrektur des erzeugten Modells. Die Unterstützung iterativer Entwicklung auf Basis der Petmap stellt einen weiteren wesentlichen Mehrwert gegenüber anderen Arbeiten dar.

7 Auswertung und zukünftige Herausforderungen

Die Integration von Movisa in den Entwicklungsprozess der Useware erforderte einerseits die Überwindung der Abstraktionslücke und zum anderen die Verwendung von Modellen in iterativen Entwicklungsprozessen. Als Lösung wird ein Ansatz mit interaktiver Transformation in Ergänzung mit der Petmap, ein Mittel zur persistenten Speicherung von Mappings, vorgestellt. Die Interaktionen beziehen den Entwickler der Benutzungsschnitt-

stelle direkt in die Transformation ein, sodass erwartungskonforme Elemente erzeugt und die manuelle Nachbearbeitung überflüssig werden. Die adäquate Wahl der Elemente ist dem Entwickler überlassen, der bei Verwendung von Movisa ein Experte der Automatisierungstechnik ist und somit über das notwendige Domänenfachwissen verfügt. Die interaktive Transformation ist ein vielversprechender Ansatz zur Lösung des Mapping-Problems, der nicht auf die hier verwendeten Metamodelle oder die MBUID beschränkt ist, sondern der ein allgemeingültiger Lösungsansatz für die Überbrückung von Abstraktionslücken bei Modelltransformationen ist.

Die iterative Entwicklung wird durch den vorgestellten Ansatz unterstützt, jedoch werden noch nicht alle Aspekte berücksichtigt, denn das in Abbildung 8 dargestellte Vorgehen realisiert nur die Veränderungen des übergeordneten DISL-Modells, die durch Transformation unter Verwendung bereits vorgenommener Mappings in ein Movisa-Modell überführt werden. Der Useware-Entwicklungsprozess sieht aber Nutzerevaluationen in allen Entwicklungsphasen vor. Für eine vollständige Integration im Sinne der Definition müssen auch Veränderungen auf Movisa-Ebene gemäß den Ergebnissen der Nutzerevaluation vorgenommen werden können. Auf diese Weise können *Modellinkonsistenzen* entstehen, da beide Modelle unabhängigen Veränderungen unterliegen. Die Petmap ist mit den Mappings der durchgeführten Transformationen ein mögliches Mittel zur Erkennung von Inkonsistenzen zwischen Modellen. Es ist daher zu untersuchen, inwieweit die Mappings der Petmap genutzt werden können, um direkt auf Movisa-Ebene Nutzerfeedback zu berücksichtigen. Mit entsprechenden Mechanismen (z. B. Wizards) in den Movisa-Editoren können kleine Veränderungen konform zu den Mappings (Wechsel der UIComponents innerhalb der Zuordnungsoptionen) ermöglicht werden. Dieses Vorgehen schränkt jedoch die Möglichkeiten der Benutzungsschnittstellengestaltung erheblich ein. Alternativ können beliebige Veränderungen und somit kurzzeitige Modellinkonsistenzen erlaubt werden, die durch Rücktransformationen und unter Verwendung einer entsprechenden Logik aufgelöst werden.

Die genannten zukünftigen Herausforderungen überführen das iterative Vorgehen zu einem vollständigen Roundtrip-Engineering, das eine interessante Problemstellung für die modellbasierte Entwicklung von Benutzungsschnittstellen darstellt. Auf diesem Gebiet konnten wir erste Erfahrungen sammeln und werden diese in weiteren Arbeiten vertiefen.

Literatur

[Aqu09] N. Aquino. Adding flexibility in the model-driven engineering of user interfaces. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, 2009.

[BH07] Annerose Braune und Stefan Henning. Technologieunabhängiges HMI-Engineering für technische Prozesse. In *VDI-Berichte GMA-Kongress in Baden-Baden*, 2007.

[CH03] K. Czarnecki und S. Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.

ISBN 978-3-88579-286-4

[CLC04] T. Clerckx, K. Luyten und K. Coninx. The mapping problem back and forth: customizing dynamic models while preserving consistency. In *Proceedings of the 3rd annual conference on Task models and diagrams*, 2004.

[GPR06] Volker Gruhn, Daniel Pieper und Carsten Röttgers. *MDA: Effektives Software-Engineering mit UML2 und Eclipse*. Springer, 2006.

[HB11] Stefan Hennig und Annerose Braune. Sustainable Visualization Solutions in Industrial Automation with Movisa—a Case Study. *IEEE Conference on Industrial Informatics*, 2011.

[HdBLB11] Stefan Hennig, Jan Van den Bergh, Kris Luyten und Annerose Braune. User Driven Evolution of User Interface Models — the FLEPR Approach. *Interact*, 2011.

[HKB10] St. Hennig, E. Koycheva und A. Braune. Domänenspezifische Sprachen und deren Bedeutung für die modellgetriebene Softwareentwicklung in der Automatisierungstechnik. In *11. Branchentreff der Mess- und Automatisierungstechnik*, VDI-Berichte/VDI-Tagungsbände, Baden-Baden, 2010.

[HSL+08] J. Helms, R. Schaefer, K. Luyten, J. Vermeulen und M. Abrams. User Interface Markup Language (UIML) Version 4.0. *OASIS Standard Specification, OASIS*, 2008.

[LV04] Quentin Limbourg und Jean Vanderdonckt. Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proceedings of the tenth ACM international conference on Multimedia*, 2004.

[Mei10] G. Meixner. *Entwicklung einer modellbasierten Architektur für multimodale Benutzungsschnittstellen*. Dissertation, TU Kaiserslautern, 2010.

[MGS08] G. Meixner, D. Görlich und R. Schäfer. Unterstützung des Useware-Engineering Prozesses durch den Einsatz einer modellbasierten Werkzeugkette. VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik, 2008.

[PE99] A. Puerta und J. Eisenstein. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems*, 1999.

[PSS09] Fabio Paterno', Carmen Santoro und Lucio Davide Spano. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction*, 16, 2009.

[Ran10] D. Raneburger. Interactive model driven graphical user interface generation. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, 2010.

[Sch07] Robbie Schäfer. *Model-based Development of Multimodal and Multi-device User Interfaces in Context-aware Environments (C-LAB Publication)*. Dissertation, Universität Paderborn, 2007.

[Sei09] M. Seißler. Automatisierte Transformation von Aufgabenmodellen in Dialogmodelle am Beispiel der Modellierungssprache useML 2.0, 2009.

[SS08] M. P Siikarla und T. J Systa. Decision reuse in an interactive model transformation. In *Software Maintenance and Reengineering*, 2008.

[Van05] J. Vanderdonckt. A MDA-compliant environment for developing user interfaces of information systems. In *Advanced Information Systems Engineering*, 2005.

[Zü04] D. Zühlke. *Useware-Engineering für technische Systeme*. Springer, 2004.

ISBN 978-3-88579-286-4